실시간 시스템 검증을 위한 지역모형검사

박재호[†] · 김성길[†] · 황선호[††] · 김성운[†††]

요 약

실시간 검증은 명세와 요구사항과의 논리적 정확성 뿐만 아니라 시간적 정확성을 확인하는 일련의 과정이다. 하지만 시간의 무한성에 의해 시스템 상태가 무한히 증가할 수 있는 상태 폭발 문제가 검증과정에서 중요한 문제점이 되고 있다. 본 논문에서는 형식 검증에 기반을 두며, 시스템의 행위 측면을 시간 오토마타로 기술한 시스템 모델이 Timed mu-calculus로 표현된 시스템의 특성에 만족하는지의 여부를 통해 명세의 완전성을 확인하는 실시간 검증 기법을 기술한다. 이를 위해 초기상태의 논리값에 초점을 두어 검증과정에서 필요로 하는 노드로만 Product Graph를 구성하여 노드 값을 결정해 나가는 지역모형검사 기법에 대해 제안한다. 이 방법은 모델의 모든 상태를 조사하지 않으므로 상태 폭발 문제를 최소화 시킬 수 있어 실시간 시스템 검증에 효과적으로 적용이 가능하다.

Local Model Checking for Verification of Real-Time Systems

Jaeho Park[†], Seongkil Kim[†], Sunho Hwang[††] and Sungun Kim[†††]

ABSTRACT

Real-Time verification is a procedure that verifies the correctness of specification related to requirement in time as well as in logic. One serious problem encountered in the verification task is that the state space grows exponentially owing to the unboundedness of time, which is termed the state space explosion problem. In this paper, we propose a real-time verification technique checking the correctness of specification by showing that a system model described in timed automata is equivalent to the characteristic of system property specified in timed modal-mu calculus. For this, we propose a local model checking method based on the value of the formula in initial state with constructing product graph concerned to only the nodes needed for verification process. Since this method does not search for every state of system model, the state space is reduced drastically so that the proposed method can be applied effectively to real-time system verification.

1. 서 론

최근에 비디오나 사운드 같은 연속적인 미디어가 컴퓨터 네트워크에 적용되기 시작하면서 미디어에 따른 시간적 특성을 보장할 수 있는 네트워크와 프로토콜 등이 요구되고 이를 만족시킬 수 있는 실시간 시스템들이 활발히 개발되고 있다. 실시간 시스템은 기존의 컴퓨터 시스템과 달리 시스템 동작의 정확성이 논리적 정확성 뿐만 아니라 시간의 경과, 지연, 제약, 재설정 등과 같은 시간의 정확성에도 좌우되는 시스템을 말한다. 즉, 작업의 논리적 결과뿐만 아니라 그 결과가 생성된 시간에 따라 정확성이 결정되는 시스템으로 물리적인 주변환경에서 입력을 받아 제어 작업을 수행하여 출력을 내는 과정에 적시성이 부여되기 때문에 데드라인이라는 어떤 시간 한계를 넘지 않고 논리적으로 정확한 출력이 보장되어야 예기치 않은 결과를 피할 수 있게 된다.

그런데 실시간 시스템의 개발과정에서는 실제로 구현하기 전에 명세가 요구사항의 시간 제약 조건을 만족하고 높은 신뢰성을 보장할 수 있는지에 대한

본 연구는 부경대학교 기성회 연구지원에 의해 수행되었음.

† 부경대학교 정보통신공학과
†† 한국전파기지국관리(주) 연구소장
††† 부경대학교 정보통신공학과 조교수

확인이 필요하다. 이와 같은 일련의 과정을 실시간 검증이라 하고, 대표적인 검증 기법으로 시뮬레이션과 형식 검증 등이 있다. 먼저 실시간 시뮬레이션은 실제 시스템의 기능을 입력과 시간에 대한 함수로 수학적으로 모델링하고, 이를 프로그램으로 구현하여 실행시킨 후 그 결과를 분석하는 과정이다. 이러한 실시간 시뮬레이션 기법은 시간적 특성에 따른 변화에 대한 요구사항을 파악하기 쉽기 때문에 많은 개발자들이 사용하고 있으나, 검증 시간이 오래 걸리고 결과가 복잡하여 분석이 용이하지 않다는 단점이 있다.

반면에 실시간 형식 검증 기법은 시스템의 행위 측면을 TA(Timed Automata), TLTS(Timed Labeled Transition System) 등의 시스템 명세 언어(System Specification Language)로 기술한 시스템 모델이 TCTL(Timed Computation Tree Logic)이나 Timed mu-calculus와 같은 임의의 특성을 표현한 시스템 특성 언어(System Property Language)로 기술된 수학적인 논리식을 만족하는지의 여부를 통해 시스템 명세의 완전성을 확인한다. 이는 시스템의 시간에 따른 행위 측면만을 검사하는 제한적 범위를 가지지만, 시뮬레이션 기법에 비해 검증시간이 단축되고 결과 분석이 용이하며 시스템의 특성을 매우 간결하고 명확하게 표현하고 자동화가 가능하다는 장점이 있다.

대표적인 형식 검증 기법으로 통신 프로토콜이나 하드웨어 설계시 검증 기법으로 활용되어 왔던 모형검사를 들 수 있는데[1], 이는 검증 결과의 유도 과정에 따라 전역모형검사와 지역모형검사로 나뉠 수 있다. 전역모형검사는 검증시 시스템 명세 언어의 모든 상태와 시스템 특성 언어의 모든 변수의 짝으로 이뤄진 노드를 모두 탐색함으로써 검증 결과를 유도해낸다. 하지만 실시간 시스템에서는 시간의 무한성으로 인해 시스템 내의 상태가 무한히 증가할 수 있으므로 컴퓨터를 통한 검증 과정에서 메모리 초과 등과 같은 문제가 초래될 수 있다. 따라서 검증시 불필요한 노드의 탐색을 억제하여 원하는 명제의 완전성 여부를 확인하는 지역모형검사가 요구된다.

따라서 본 논문에서는 실시간 시스템의 검증을 위해 시스템 모델의 필요한 상태만을 탐색하여 컴퓨터 자원의 절감과 검증 시간을 단축시킬 수 있는 지역모형검사 기법을 제안한다. 이를 위해 본 논문의 2장에서는 비실시간 시스템의 논리적 정확성 검증을 위한 지역모형검사 기법에 대해 기술하고, 3장에서는 논리적 정확성 뿐만 아니라 시간적 정확성까지 확인할 수 있는 실시간 지역모형검사 기법에 대해 제안한다. 마지막으로 4장에서는 결과와 향후 연구사항에 대해 기술한다.

2. 비실시간 시스템의 검증

본 장에서는 비실시간 검증을 위해 시스템의 행위와 특성을 표현할 수 있는 정형언어인 LTS(Labeled Transition System)와 M_μ(Modal mu-calculus)에 대해 정의한다. 또, 검증시 필요한 시스템 모델의 상태와 논리변수만을 탐색하는 지역모형검사 기법인 Explore 알고리즘에 대해 기술한다.

2.1 Labeled Transition System

시스템의 행위를 기술하는 시스템 명세 언어로 사용되는 LTS는 비실시간 지역모형검사를 위한 시스템 모델을 제공하게 되며, 다음과 같이 4-tuple로써 구성된다[2,3].

$L=\langle S, s_0, Act, Tr\rangle$
S: 상태의 집합
s_0: 시작 상태, $s_0 \in S$
Act: 행위의 집합
Tr: 천이의 집합, $Tr \subseteq S\times\Sigma\times S$, $tr\in Tr$, $tr=\langle s, a, s'\rangle$

LTS를 L이라 하면 S는 시스템이 가질 수 있는 상태들의 집합을 나타내며, s_0는 LTS가 시작되는 상태를 나타낸다. 행위 집합 Act는 한 상태에서 다른 상태로 천이를 유발시킬 수 있는 행위들의 집합으로써 입력행위, 출력 행위 혹은 두 행위가 공존하는 입/출력 쌍이 Act의 원소가 될 수 있다. 천이집합 Tr은 특정 천이 함수를 사용하여 상태를 바꾸는 요인이 되는데 위의 정의식에서처럼 천이 함수는 상태와 심볼 그리고 다음 상태의 쌍으로 구성된다.

2.2 Modal mu-calculus

형식 논리(Modal logic)는 어떤 시스템의 각 상태에서 참(true)을 찾아가는 형태를 나타내는 "<> (possibility)"와 "[] (necessity)"의 두가지 연산자를 사용하여 시스템의 특성을 기술하는 논리식이다. 그

리고 시제 논리는 형식 논리의 두가지 연산자를 시제 관점에서 바라본 것으로 두 연산자는 각각 "<> (eventually)"와 "[] (always)"로 해석되어 특성을 기술할 수 있다. 또한 mu-calculus는 minimization operator(mu 또는 μ)인 greatest fixed point(ν)와 least fixed point(μ)를 사용하여 주어진 시스템 상태의 참과 거짓 여부를 확인하는 논리 계산법으로, greatest fixed point는 모든 상태를 초기에 참으로 설정하여 거짓을 찾아가게 되고 반대로 least fixed point는 모든 상태를 초기에 거짓으로 설정하여 참을 찾아간다.

Modal mu-calculus는 형식논리의 변형인 시제 논리를 사용하여 시스템의 특성을 기술하고 fixed point 계산법을 사용하여 참과 거짓을 찾아가는 논리로써 비실시간 지역모형검사에서 시스템 특성을 나타내는 논리식으로 사용된다. 일반화된 Modal mu-calculus의 논리식은 아래와 같다[4,5].

$$\Phi ::= tt \mid ff \mid p \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [k]\Phi \mid \langle k \rangle \Phi \mid \nu Z.\Phi \mid \mu Z.\Phi \mid$$

"Φ"는 시스템의 특성을 나타내는 논리식으로 참과 거짓의 값을 가진다. 극소명제 "*tt*"와 "*ff*"는 항상 참과 거짓임을 각각 나타내며, "*p*"는 임의의 상수적인 명제를 나타낸다. 논리연산자 "∧"와 "∨"는 각각 논리곱과 논리합을 가리키고, 행위연산자 "[*k*]"는 necessity를 나타내는 연산자로써 행위 k가 반드시 발생할 경우에 참이 되고, "<*k*>"는 possibility를 나타내는 연산자로써 어떠한 행위 k도 발생하지 않을 경우에 거짓이 된다. "ν"는 greatest fixed point를 나타내는 연산자로써 "ν"에 속한 모든 명제적 변수의 초기값은 참으로 설정되어 거짓인 상태를 찾게 되고, "μ"는 least fixed point를 나타내는 연산자로써 "μ"에 속한 모든 명제적 변수의 초기값을 거짓으로 설정하여 참인 상태를 찾게 된다.

2.3 비실시간 지역모형검사

2.3.1 Explore 알고리즘

지역모형검사 기법인 Explore 알고리즘[9]은 LTS 모델의 초기상태의 논리값에 초점을 두고, M_μ로 기술된 해당 논리 변수에 대해 LTS 모델의 각 상태가 만족하는지를 확인해 나가는 과정이다. 이 때, 탐색 과정과 결과 출력을 위해 Product Graph(이하 PG)가 사용되고, 노드에 관한 정보를 가지는 참조 변수 *V, N, D, U*가 사용된다. 각 변수에 대한 설명은 아래에 기술하였다.

PG는 M_μ의 논리 변수와 LTS 모델의 상태로 이뤄진 노드<변수, 상태>와 각 노드를 연결하는 에지로 구성된다. 먼저 노드는 (그림 1. a)에서와 같이 변수명과 상태명을 가지고 있으며, 노드의 값이 참인지 거짓인지를 나타내는 *Value*와 탐색되어야 할 노드의 개수를 표시하는 *Counter*, 그리고 노드의 값이 정해졌음을 알리는 *done*을 가진다. 여기서 PG 노드의 좌측 상단은 *Counter*를, 우측 상단은 *Value*를 그리고 색깔 유무는 *done*의 값을 가리킨다. 또한 노드는 인접하는 후속 노드와의 논리적 상관관계에 따라 "∨"와 "<>"는 ∨-node라 하고 "∧"와 "[]"는 ∧-node라 한다. 에지는 각 노드의 종속관계와 논리적 상관관계를 나타내는데, 만약 논리식이 $X_0=X_1 \wedge X_2$인 경우 (그림 1. b)와 같이 X_0는 두개의 후속 노드 X_1와 X_2를 가지고 논리연산자 "∧"의 관계를 가지게 된다.

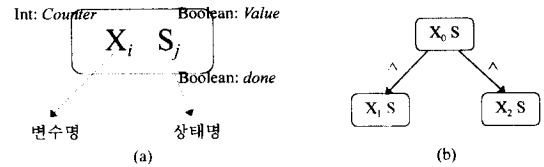



그림 1. PG 노드와 에지의 구성

지역모형검사에서는 차례로 노드가 생성되면서 PG를 구성하기 때문에 하나의 노드가 생성될 때마다 *Value*와 *Counter*의 값을 초기화시켜야 한다. *Value*의 초기화 규칙은 시스템 모델 *L*과 M_μ의 fixed point 계산법에 따라 *max* 블록(greatest fixed point "ν")에 속한 노드인 경우 true로 *min* 블록(least fixed point "μ")에 속한 노드일 경우에는 false로 값을 설정한다. *Counter*의 초기화 규칙은 ∨-node인 경우에 참인 후속노드의 개수로, ∧-node인 경우에는 거짓인 후속노드의 개수로 초기화시킨다. ∨-node의 *Counter*는 거짓인 후속노드가 나타날 때마다 감소하며, *Counter*의 값이 0일 때 더 이상 참인 후속노드가 존재하지 않으므로 노드의 최종값은 거짓이 된다. 역으로 ∧-node의 *Counter*는 참인 후속노드가 나타

날 때마다 감소하며, *Counter*의 값이 0일 때 더 이상 거짓인 후속노드가 존재하지 않으므로 노드의 최종 값은 참이 된다.

노드 값의 결정 방법은 임의의 노드 t = $<X_i, S_j>$가 후속노드가 없는 최종 노드일 경우에 X_i가 극소명제 *p*(*tt* 또는 *ff* 포함)이면, 상태 S_j가 극소명제 *p*를 만족하는지의 여부에 따라 값이 결정된다. X_i가 행위 연산자 "<*a*>" 이고 S_j가 행위 *a*를 가지고 있지 않을 경우, t의 값은 false가 되고, X_i가 행위 연산자 "[*a*]" 이고 인접하는 후속노드의 값이 모두 참일 때 t는 참이 된다. 논리 연산자("∨", "∧")의 경우 후속하는 노드의 값에 따라 t의 값이 결정된다.

참조변수 *V*, *N*, *D*, *U*는 PG 노드에 관한 정보를 가지고 노드 생성시 사용된다. 먼저 *V*(visited)는 탐색된 노드의 집합을 가지며, 주어진 노드가 탐색되었는지를 효율적으로 결정하기 위한 탐색 구조를 제공한다. *N*(located node)는 탐색되어질 노드의 집합을 가지며, 스택 구조로 되어있다. *D*(decided)는 노드의 값은 결정되었으나 전속 노드의 값이 결정되지 않은 노드를 가지게 된다. *U*(undecided)는 모든 후속 노드의 값이 탐색되었으나, 아직 값이 결정되지 않은 노드를 가지며 스택 구조로 이뤄져 있다.

이상과 같이 PG와 참조변수를 사용한 비실시간 지역모형검사를 요약하면 다음과 같다.

(1) 초기 노드 $<X_0, S_0>$를 가진 PG 생성
(2) *V*, *N*, *D*, *U* 변수 생성 및 *Explore* 알고리즘 수행
(3) $<X_0, S_0>$의 값이 결정되었으면 알고리즘 종료

LTS 모델 *L*과 M_μ 논리식 *B*를 입력으로 하는 *Explore* 알고리즘은 다음과 같다.

```
procedure Explore(L, B)
add <X0, S0> to V and N
while N ≠ NULL
  t:=fisrt(N)
  if t has no successors then        // t는 최종노드
    decide on t's value
    done(t):=true
    move t from N to D
    processD
  else if ¬done(t) then
    if t has unexplored successors then
      choose the next successor w of t
      introduce the edge t→w
      if w∉V then  // w는 탐색되지 않은 노드
        add w to V and N
      else if done(w) then
        report the value of w to t
        if done(t) then
          add t to D
          processD
    else
      move t from N to U
  else remove t form N        // t는 값이 결정
  if N is empty or Bt≠Bfirst(N) then
    processU(t)
```

현재의 *done*인 노드의 값을 가지고 전속노드의 값을 결정하게 되는 *processD*는 PG를 입력으로 하며 다음과 같은 절차로 수행된다.

```
procedure processD(PG)
 while D≠∅
   remove some t from D
   for each w such that w→t and¬done(w)
     report the value of t to w
     if done(w) then
   add w to D
```

*processU*는 *N*이 빌 경우 혹은 B_t와 $B_{first(N)}$가 서로 다를 경우에 수행되는데, 이는 후속노드가 탐색되어졌지만 아직 값을 결정하지 못한 노드의 값을 결정할 때 사용된다.

```
procedure processU(t,PG)
 while Bfirst(U)=Bt
   remove w from U
   if¬done(w) then
     done(w):=true
     add w to D
   processD
```

Explore 알고리즘에서는 어떤 노드 $<X_i, S_j>$가 *N*

에서 추출될 때, $done$<X$_i$, S$_j$>=true가 되거나 U에 위치하게 된다. 또한 U에 있는 모든 노드도 $processU$에 의해 결국은 $done$<X$_i$, S$_j$>=true가 된다. 따라서 PG의 모든 노드는 $done$이 되고, 정확하게 계산된 최종값을 가지게 된다.

2.3.2 비실시간 지역모형검사 적용 예

"항상 결과적으로 p를 만족한다"라는 명제를 M_μ의 논리식으로 표현하면 다음과 같다.

$$\nu Y.(\mu Z.p \vee [a]Z) \wedge [a]Y$$

위의 논리식에 대해 (그림 2)의 시스템 모델의 만족 여부를 Explore 알고리즘에 적용하여 확인한다.

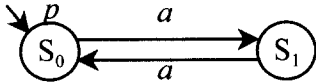



그림 2. LTS로 기술된 시스템 모델

먼저, M_μ 논리식을 fixed point 계산법에 따라 극소명제가 될 때까지 max 블록과 min 블록으로 분리한다.

$B_{max} \equiv max\{X_0 = X_1 \wedge X_2$ $X_2 = [a]X_0\}$	$B_{min} \equiv min\{X_1 = X_3 \vee X_4$ $X_3 = p$ $X_4 = [a]X_1\}$

극소명제로 분리된 각 변수와 LTS 모델의 각 상태의 짝인 노드로 구성된 최종 PG는 (그림 3)과 같다.

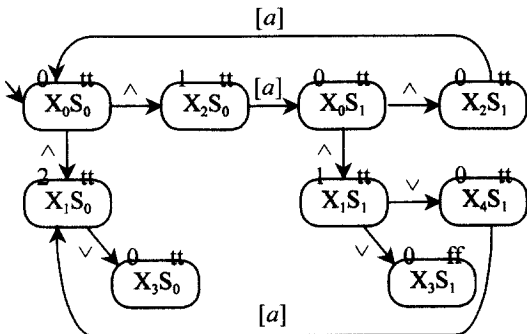



그림 3. Explore 알고리즘 적용 후 구성된 최종 PG

(그림 3)에서 보듯이 초기 노드 <X$_0$, S$_0$>의 값이 "tt"로 결정되었기 때문에 (그림 2)의 시스템 모델은 논리식을 만족하게 된다. PG 구성에 따른 설명은 3.4.2절 실시간 지역모형검사 적용 예에서 자세히 다루기로 한다.

3. 실시간 시스템의 검증

본 장에서는 실시간 시스템의 검증을 위해 실시간적 행위와 특성을 표현할 수 있는 정형언어인 시간 오토마타와 Timed mu-calculus에 대해 정의한다. 또한 2장에서 기술된 지역모형검사를 실시간 검증에 적용하기 위해서, PG에 시간 개념을 도입한 RPG (Region Product Graph)에 대해 기술하고, 실시간 지역모형검사 알고리즘을 제안한다.

3.1 시간 오토마타 (Timed automata)

실시간 시스템의 시간에 따른 천이를 기술하는 시스템 명세 언어로 사용되는 시간 오토마타는 시간 값을 가질 수 있는 변수인 클럭(clock)과 오토마타의 천이를 시간적으로 제약할 수 있는 시간제약 요소들의 조합을 통해 실시간에 따른 천이를 표현할 수 있다[6].

시간 오토마타 A_T는 다음과 같이 6-tuple로 구성된다.

A_T = $<S, S_0, \Sigma, C, Tr, F>$
S: 상태의 집합
S_0: 시작 상태의 집합
Σ: 심볼의 집합
C: 클럭의 집합
Tr: 천이의 집합, $Tr \subseteq S \times \Phi(C) \times 2^C \times \Sigma \times S, tr \in Tr$
$\lambda \in \Lambda \subseteq C,\ a \in \Sigma,\ tr = <s, \phi, a, \lambda, s'>$
F: 최종 상태의 집합

여기서 $\Phi(C)$는 시간제약의 집합이고, 2^C는 클럭의 멱집합이며, λ는 재설정되는 클럭 변수의 집합을 나타내며, 시간 제약은 "$\phi := x \le t \mid t \le x \mid x \le y \mid \sim\phi \mid \phi_1 \wedge \phi_2 \mid x, y \in C$, t:constant"인 형태를 지닌다.

3.2 Timed mu-calculus

T_μ(Timed mu-calcuslus) 는 2장에서 기술된 Modal

mu-calculus의 기본 구문 외에 미래의 어떤 시간을 명시할 수 있는 시간 제약(time constraint) 극소 명제(atomic proposition)와 시간을 시스템의 특성 상 요구되는 임의의 상수로 설정할 수 있는 시간 재설정 연산자를 가지고 있다. 또한 행위에 따른 후속 관계를 명시한 행위 연산자 “<> possibility, [] neccesity”에 상응하는 시간에 따른 후속 관계를 명시한 시제연산자 “∃ eventually, ∀ always”가 T_μ의 구문에 포함되어 있어 이를 사용하여 시스템 모델의 시간에 따른 특성을 표현할 수가 있다. T_μ의 구문은 다음과 같이 정의된다[7,8].

$$\Phi ::= tt \mid ff \mid p \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [k]\Phi \mid <k>\Phi$$
$$\mid \nu Z.\Phi \mid \mu Z.\Phi \mid \phi_t$$
$$\phi_t ::= C_t \mid x.\Phi \mid \exists \Phi \mid \forall \Phi \mid$$

여기서 ϕ_t는 시간적 특성을 기술하기 위해 추가된 시간 구문으로써 “C_t”는 시간제약을 가리키는 극소 명제이고, “x”는 x라는 클럭을 임의의 상수로 재설정하는 연산자이며, “∃”과 “∀” 시제 연산자로써 각각 미래의 어떤 시간과 미래의 모든 시간을 가리키는 연산자이다.

3.3 Region Product Graph

비실시간 지역모형검사에서 사용되었던 PG의 노드를 해당하는 시간에 따라 구분하여, <변수, 상태, 시간>의 노드들로 이루어진 Timed Product Graph (TPG)를 구성함으로써 앞 장에서 기술된 지역모형검사를 사용하여 실시간 시스템을 쉽게 검증할 수 있다. 하지만 불행하게도 시간 단위가 실수배로 증가하는 밀집 시간의 경우에 어떤 시간영역 내에 존재하는 시간이란 측정할 수 없이 무한하게 됨으로써, 시스템 모델의 상태, 논리식의 변수 그리고 시간의 짝으로 구성되는 TPG의 노드도 시간의 무한성에 따라 무한히 증가하게 된다. 이를 해결하기 위해 본 논문에서는 시간을 어떤 공통된 영역으로 묶어줌으로써 노드 개수를 최소화 시키는 RPG를 이용한 지역모형검사 기법을 제안한다.

RPG는 TPG에서 발전되었기 때문에 아래의 기본 TPG 천이 규칙(①~④)과 전제조건(⑤,⑥)을 만족한다.

① TPG 노드는 <변수, 상태, 시간값>인 3-tuple로 구성

$$< X, s, \pi > \xrightarrow{operator} < X', s', \pi' >$$

② 논리연산자에 의한 천이시 변수는 변하지만 상태와 시간은 불변

$$< X, s, \pi > \xrightarrow{\vee,\wedge} < X', s, \pi >$$

③ 행위연산자에 의한 천이시 변수와 상태는 변하지만 시간은 불변 (단, s에서 s′로 향하는 act가 존재)

$$< X, s, \pi > \xrightarrow{<act>,[act]} < X', s', \pi >$$

④ 시제연산자에 의한 천이시 변수와 시간은 변하지만 상태는 불변

$$< X, s, \pi > \xrightarrow{\exists,\forall} < X', s, \pi' >$$

⑤ 모든 $< X, s, \pi > \in < X', s', \rho >$는 $< X', s', \rho' >$에 후속하는 TPG 노드를 가지고 있다.

⑥ $< X, s, \rho >$에 있는 모든 TPG 노드의 명제값은 모두 동일하다.

⑤는 RPG 가 항상 일관성 있게 구성됨을 나타내고, ⑥은 RPG 노드를 PG 노드에서와 같이 하나의 명제값을 가지는 극소 개체로 간주함을 나타낸다.

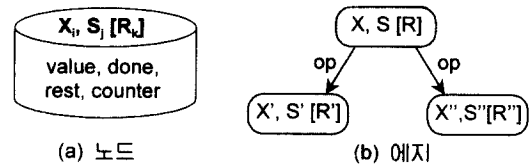



그림 4. Region Product Graph

(그림 4)에서 보듯이 RPG는 유한개의 노드와 각 노드를 연결하는 에지로 구성된 단방향 그래프이다. 노드는 Timed mu-calculus로 표현된 논리 변수 X_i와 시간 오토마타로 기술된 실시간 모델의 상태 S_j 그리고 해당 노드가 존재할 수 있는 시간의 범위를 나타내는 시간영역 R_k의 쌍으로 구성된 인식자로써 다른 노드들과 구분된다. 이 때 R_k는 실수의 범위를 가지는 밀집시간을 시간단위로 사용한다. 각 노드는 인식자 이외에도 노드의 특성을 나타내는 변수들과 0 이상의 후속노드와 연결되는 에지들을 가진다. 변수는 value, done, rest, counter로 구성된다. value는 노드가 참인지 거짓인지를 나타내는 변수로써 Timed mu-calculus의 fixed point 형태에 따라 greatest fixed point에 속하면 참으로, least fixed point에 속

하면 거짓으로 초기화되며, 후속노드의 value에 따라 값이 변경될 수 있다. done은 value 값이 최종 결정되어 값이 바뀔 수 없음을 나타낸다. rest는 노드 생성시 연산자에 따라 최대로 가질 수 있는 후속노드의 인식 요소들을 가지는 집합이다. 즉, 논리연산자일 때는 T_{μ}의 논리변수, 행위연산자일 경우에는 모델의 상태, 시제연산자일 때는 노드가 가질 수 있는 시간영역의 합집합을 가지게 되며, 그 외의 연산자일 경우에는 후속노드가 존재하지 않는 경우로써 rest는 공집합이 된다. RPG에 후속노드가 하나씩 추가될 때마다 rest 중 하나가 추출되어 후속노드의 인식자에 전달된다. counter는 노드가 가질 수 있는 후속노드의 최대 개수로써 초기화되고 값이 결정된 후속노드가 있을 때마다 1씩 감소된다. 에지는 노드와 그 후속노드들과 연결되어 Timed mu-calculus의 연산자(op)에 따른 상관관계를 나타낸다.

RPG는 실시간 지역모형검사 알고리즘에 따라 생성되고 초기노드가 값이 결정되면 RPG의 구성은 종료된다. 최종적으로 구성된 RPG는 실시간 안전성 검증 및 실시간 필연성 검증을 위해 제공된다.

3.4 실시간 지역모형검사

3.4.1 실시간 지역모형검사 알고리즘

(그림 5)는 본 논문에서 제안하는 시간 오토마타 명세 모델과 Timed mu-calculus 논리식을 입력으로 하는 실시간 지역모형검사 알고리즘이다. 먼저 초기화를 통해 실시간 안전성 또는 실시간 필연성을 fixed point 계산법에 따라 max 블럭과 min 블럭으로 나누고, RPG의 초기노드를 생성하며, RPG의 노드 정보를 가지는 스택 구조의 참조변수 V, N, D, U, HC를 초기화한다. V는 RPG에서 이미 탐색된 노드들의 집합이며, N은 탐색되어질 노드들의 집합, D는 노드값이 결정되었지만 전속노드의 값이 결정되지 않은 노드들의 집합이고, U는 모든 후속노드의 값이 탐색되었지만, 아직 값이 결정되지 않은 노드들의 집합이다. HC는 실시간 필연성 검증시 T_{μ}에서 사용된 논리식과 시간 오토마타에서 사용된 클럭이 다를 경우 T_{μ} 클럭과 시간 오토마타 클럭과의 변환 관계를 나타낸다. 초기화가 완료되면, RT-Explorer에서 초기노드의 값이 결정될 때까지 ProcessRest, DecideValue, ProcessD, ProcessU 프로세스 호출과 참조변수 V, N, D, U, HC의 삽입/추출 과정을 통하여 RPG를 구성하고 각 노드의 변수를 결정한다. RPG 구성이 완료된 후, RPG를 입력으로 검사를 통해서 명세 모델이 원하는 검증 항목에 맞는지를 확인한다. 초기노드의 값이 참이면 명세 모델이 해당 검증 항목을 만족하는 것이고, 그렇지 않으면 만족하지 않는 것이다.

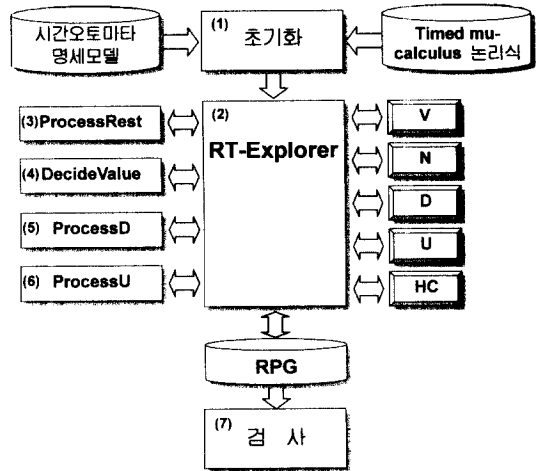



그림 5. 실시간 지역모형검사 알고리즘 구성도

(1) 초기화

시간 오토마타 명세 모델과 실시간 안전성 논리식 또는 실시간 필연성 논리식을 입력 받아, 실시간 지역모형검사 알고리즘의 초기화를 수행하는 과정으로 블럭초기화, RPG 초기화, 참조변수 초기화로 나뉜다.

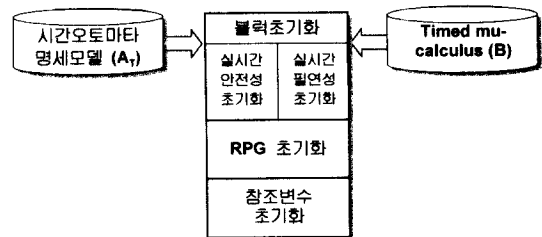



그림 6. 실시간 지역모형검사 알고리즘의 초기화 과정

블럭초기화는 사용자로부터 원하는 검증 대상인 시간 오토마타 명세 모델과 검증 항목을 입력받아, T_{μ} 논리식의 각각의 명제적 변수를 fixed point 계산법에 따라 max 블럭과 min 블럭으로 나눈다. 명제적

변수가 greatest fixed point에 속하면 max 블럭으로, least fixed point에 속하면 min 블럭으로 나누어 논리식을 블럭화 한다.

명세 모델 내의 임의의 상태에서 논리적, 시간적 정확성이 결여되어 천이가 발생하지 않는 timed deadlock이나 상태의 부분집합만을 무한히 반복하는 timed livelock의 존재를 확인하기 위한 실시간 안전성 T_μ 논리식은 "$\nu Y.\forall[-]Y \wedge (\mu X.p \vee \exists\langle - \rangle X)$"이다. 이 때 극소명제 p는 초기상태만이 만족한다. 다음은 실시간 안전성 초기화 과정을 통해 블럭화된 실시간 안전성 T_μ 논리식 $B(B_{max}+B_{min})$이다.

$B_{max} \equiv max\{X_0 = X_1 \wedge X_2$	$B_{min} \equiv min\{X_2 = X_4 \vee X_5$
$X_1 = \forall X_3$	$X_4 = p$
$X_3 = [-]X_0\}$	$X_5 = \exists X_6$
	$X_6 = \langle - \rangle X_2\}$

명세 모델 내에서 행위 *act1*이 발생한 후 시간제약 *timec*를 만족하면서 행위 *act2*가 발생하는지를 확인하기 위한 실시간 필연성 T_μ 논리식은 "$\nu Y.\forall[act1]z.(\mu X.\exists\langle act2\rangle timec \vee \exists\langle - \rangle X) \vee \forall[-]Y$"이다. 다음은 사용자로부터 *act1*, *act2*, *timec*를 입력받아 실시간 필연성 초기화 과정을 통해 블럭화된 실시간 필연성 T_μ 논리식 $B(B_{max}+B_{min})$이다.

$B_{max} \equiv max\{X_0 = X_1 \vee X_2$	$B_{min} \equiv min\{X_6 = X_7 \vee X_8$
$X_1 = \forall X_3$	$X_7 = \exists X_9$
$X_2 = \forall X_4$	$X_9 = \langle act2\rangle X_{10}$
$X_3 = [act1]X_5$	$X_{10} = timec$
$X_4 = [-]X_0$	$X_8 = \exists X_{11}$
$X_5 = z.X_6\}$	$X_{11} = \langle - \rangle X_6\}$

B의 초기변수 X_0, A_T의 초기상태 S_0 그리고 클럭 t가 0인 시간영역 R_0의 쌍으로 이루어진 초기노드를 생성하여 RPG를 초기화하고 참조변수 V, N, D, U, HC를 어떠한 노드도 가지지 않은 공(empty)의 상태로 초기화한다.

(2) RT-Explorer 프로세스

(입력) 시간 오토마타 명세 모델 A_T, T_μ의 블럭화된 논리식 B

(출력) 최종 RPG

```
process RT-Explore(A_T, B)
  t0를 V와 N에 추가
  while N≠NULL ........................................①
    t:=fisrt(N)      // N의 첫번째 노드를 t에 입력
    ProcessRest(t)
    if t가 최종노드인 경우 then ..................②
      DecideValue(t)
      done(t):=true  // t의 값이 결정되었으므로
      done은 참t를 N에서 D로 이동
      ProcessD
    else if¬done(t) then ..............................③
      if t에서 탐색되지 않은 후속노드가 존재 then
      t의 후속노드 w를 선택
      if w가 V에 존재하지 않는 경우 then
        w를 V, N과 RPG에 추가
      t→w로 에지를 연결
    else        // t의 모든 후속노드가 탐색됨
      t를 N에서 U로 이동
  else .........................................................④
      t를 N에서 제거    // t의 값이 결정된 경우
  if N이 empty or B_t≠B_first(N) then ..........⑤
  // N이 비어있거나 t의 블럭과 N의 첫번째 노드의
  블럭이 다를 경우
    ProcessU(t)
  return RPG
end process
```

RT-Explorer 프로세스는 시간 오토마타 명세 모델 A_T와 T_μ의 블럭화된 논리식 B를 입력받아 초기노드 *t0*의 값이 결정될 때까지 ProcessRest, DecideValue, ProcessD, ProcessU 프로세스 호출과 참조변수 V, N, D, U, HC의 삽입/추출 과정을 통해 RPG를 구성한다.

먼저, 초기노드 *t0*를 V와 N에 삽입하여 N이 빌 때까지 ①을 수행한다. ①에서는 N에 있는 첫번째 노드를 t에 대입한 후, t가 가질 수 있는 후속노드의 인식요소를 정하기 위해 ProcessRest(t)를 호출하여 rest(t)를 설정한 후, rest(t), done(t)와 t의 에지 유무에 따라 ②, ③, ④를 수행한다. ②는 t가 후속노드가

없는 최종노드인 경우로써, DecideValue(t)에 따라 t의 value를 결정하고 done(t)=true로 설정한 후, t를 N에서 D로 이동하여 ProcessD를 호출한 후 ⑤를 수행한다. ③은 후속노드가 존재하는 t의 값이 미결정 노드인 done(t)≠true 경우로써, t에서 탐색되지 않은 후속노드가 존재하는지를 확인한다. 만약 t에서 미탐색된 노드가 존재한다면, rest(t)에 따라 후속노드 w를 선택한다. 이 때 w가 V에 없는 노드라면 w를 V, N 및 RPG에 추가하여 t에서 w로 에지를 연결하고, w가 V에 있는 노드라면 t에서 w로 에지만 연결한다. 만약 t에서 탐색되지 않은 후속노드가 없는 경우 t를 N에서 U로 이동한다. 그리고 나서 ⑤를 수행한다. t가 ②와 ③에 해당되지 않을 경우 ④가 수행되는데, 이때는 t의 값이 결정된 경우이기 때문에 N에서 t를 제거한 후 ⑤를 수행한다. ⑤에서는 N이 비여있거나 t가 속한 블럭과 N의 첫번째 노드의 블럭을 확인하여 서로 다를 경우에 ProcessU(t)를 수행한다. N이 스택 구조로 되어있기 때문에 처음 N에 들어갔던 초기노드 t0의 값이 결정되면 N은 비게 되므로, ①을 종료하고 최종 RPG를 출력한다.

(3) ProcessRest 프로세스
(입력) rest가 설정되지 않은 노드 t = <X_i, S [R]>
(출력) rest가 설정된 노드 t = <X_i, S [R]>

---

process *ProcessRest*(*t*=<X_i, S [*R*]>)
  switch *B*의 X_i→operator
    case "∀" or "∃" .... ①
      if *R*이 empty 또는 재설정된 경우 **then**
        rest(*t*) := S의 모든 시간제약의 합집합
      else
        rest(*t*) := *R*과 S의 모든 시간제약과의 합집합
    case "[*a*]" or "<*a*>" .... ②
      if *R*이 A_T의 S에서 *a*에 의한 시간제약을 만족**then** *a*에 의한 S→S′
        rest(*t*) := S′
      else
        rest(*t*) := NULL
    case "∧" or "∨" .... ③
      X_i→X_j, X_i→X_k
      rest(*t*) := X_j와 X_k
    case "시간제약 극소명제" .... ④
      X_i의 시간제약 극소명제를 *R*과 HC에 따라 변경
      rest(*t*) := NULL
    case "reset" .... ⑤
      *R*을 HC에 추가
      X_i→X_j
      rest(*t*) := X_j
    default .... ⑥
      rest(*t*) := NULL
  return *t*
end process

ProcessRest(t) 프로세스는 후속노드의 인식요소를 위한 rest 설정을 위해서 노드 t = <Xi, S [R]>를 입력받아, rest(t)를 Xi의 연산자에 따라 설정한다. rest(t)는 RT-Explorer에서 탐색되지 않은 후속노드를 선택하여 RPG에 추가할 때 사용되는 노드 변수이다.

논리변수 Xi에 해당하는 연산자의 형태에 따라 ①, ②, ③, ④, ⑤를 각각 수행한다. ①은 연산자가 시제연산자인 경우로써, 만약 시간영역 R이 비어있거나 재설정되었다면(R=0), 상태 S에서 천이할 수 있는 모든 시간제약들의 합집합으로 rest(t)를 설정한다. 그렇지 않다면 시간영역 R과 S에서 천이할 수 있는 모든 시간제약들과의 합집합으로 rest(t)를 설정한다. ②는 연산자가 행위연산자인 경우로써, 만약 시간영역 R이 행위 *a*에 의한 천이의 시간제약을 만족한다면 a에 의해 천이가 이루어지는 상태 S′로 rest(t)를 입력한다. 그렇지 않다면 rest(t)를 NULL로 설정한다. ③은 연산자가 논리연산자이기 때문에 항상 두개의 후속노드를 가지게 되므로 Xi의 우변에 있는 논리변수 Xj와 Xk를 rest(t)에 입력한다. ④는 연산자가 시간제약 극소명제이므로, Xi의 시간제약 극소명제를 R의 클럭과 HC에 있는 클럭간의 시간관계에 따라 변경하고, rest(t)를 NULL로 설정한다. ⑤는 연산자가 시간재설정 연산자이기 때문에, R을 HC에 추가하고 Xi의 우변에 있는 논리변수 Xj를 rest(t)에 입력한다. 그 외의 연산사에 대해서는 rest(t)를 NULL로 설정한다⑥.

(4) DecideValue 프로세스

(입력) value값이 결정되지 않은 노드 t = <X_i, S [R]>

(출력) value 값이 결정된 노드 t = <X_i, S [R]>

```
process DecideValue(t=<Xi, S [R]>)
  switch B의 Xi→operator
   case "∧-노드"인 연산자 ....................... ①
     if t의 후속노드가 존재하지 않을 경우 then
       value(t) := false
     else if value가 거짓인 t의 후속노드가 존재
     then
       value(t) := false
       else if 1 감소시킨 counter(t)가 0 then
         value(t) := true
   case "∨-노드"인 연산자 ....................... ②
     if t의 후속노드가 존재하지 않을 경우 then
       value(t) := false
     else if value가 참인 t의 후속노드가 존재
     then
       value(t) := true
     else if 1 감소시킨 counter(t)가 0 then
       value(t) := false
   case "극소명제" ....................................... ③
     if t가 극소명제를 만족 then
       value(t) := true
     else
       value(t) := false
   case "시간제약 극소명제" ....................... ④
     Ct := Xi의 시간제약, w→t
       // w는 t의 전속노드
     if X(w)→operator = "∧-노드" 연산자 then
       if R이 Ct에 포함 then
         value(t) := true
       else
         value(t) := false
     else if X(w)→operator = "∨-노드" 연산자
     then
       if R이 Ct간에 교집합이 존재 then
         value(t) := true
       else
         value(t) := false
   case "reset" ............................................. ⑤
     value(t) := 후속노드의 value
  return t
end process
```

DecideValue(t) 프로세스는 value(t)를 결정하기 위해서 노드 t = <Xi, S [R]>를 입력받아, Xi의 연산자에 따라 값이 결정된 노드 t를 출력한다. 이 때 연산자가 "∧", "∀", "[]"인 노드를 "∧-노드"라 하고, "∨", "∃", "<>"인 노드를 "∨-노드"라 한다.

논리변수 Xi에 해당하는 연산자의 형태에 따라 ①, ②, ③, ④, ⑤를 각각 수행한다. ①은 t가 "∧-노드"인 경우로써, 만약 t의 후속노드가 존재하지 않거나, value가 거짓인 t의 후속노드가 발견되면, value(t)를 거짓으로 설정한다. 그렇지 않을 경우에 counter(t)를 1 감소시켜 그 값이 0이 되면 모든 후속노드에 거짓인 노드가 존재하지 않기 때문에 value(t)는 참으로 설정한다. ②는 t가 "∨-노드"인 경우로써, 만약 t의 후속노드가 존재하지 않거나, value가 참인 t의 후속노드가 발견되면, value(t)를 참으로 설정한다. 그렇지 않을 경우에는 counter(t)를 1 감소시켜 그 값이 0이 되면 모든 후속노드에 참인 노드가 존재하지 않기 때문에 value(t)는 거짓으로 설정한다. ③은 Xi의 연산자가 극소명제인 경우로써, t가 극소명제를 만족하면 value(t)를 참으로, 그렇지 않다면 거짓으로 설정한다. ④는 Xi의 연산자가 시간제약 극소명제인 경우로써, 먼저 C_t에 Xi가 나타내는 시간제약을 입력하고, t의 전속노드인 w를 찾는다. 만약 전속노드 w가 "∧-노드"이면서 시간영역 R이 C_t에 포함되면, value(t)를 참으로 설정하고, 그렇지 않다면 value(t)를 거짓으로 설정한다. 전속노드 w가 "∨-노드"이면서 시간영역 R과 C_t간에 교집합이 존재한다면, value(t)를 참으로 설정하고, 그렇지 않다면 value(t)를 거짓으로 설정한다. ⑤는 Xi의 연산자가 시간재설정 연산자인 경우로써, t의 후속노드 value를 value(t)의 값으로 설정한다.

(5) ProcessD 프로세스

```
process ProcessD
  while D ≠ NULL ...................................... ①
```

```
    D에서 t를 추출
      for t의 전속노드인 모든 w에 대해서 then
      ....................................................②
        if¬done(w) then
          DecideValue(w)
            if done(w) then
              w를 N에서 D로 이동
end process
```

ProcessD 프로세스는 노드의 값이 결정되었지만, 그 전속노드의 값이 결정되지 않은 노드를 참조변수 D에서 추출하여 전속노드의 값을 설정하기 위해 사용된다. 참조변수 D가 빌 때까지 계속 수행하게 되는 ①에서는 D에 있는 노드 t를 추출하여 ②를 수행한다. ②는 t의 전속노드인 모든 w에 대해서 done(w)가 거짓이라면 DecideValue(t)를 호출하여 값을 설정한 후, done(w)가 참이 되면 w를 N에서 D로 이동하여 ①을 반복한다.

(6) ProcessU 프로세스
(입력) RT-Explorer에서의 현재노드 t = <X_i, S [R]>

```
process ProcessU(t)
  while B_first(U)=B_t이고 U≠NULL ..............①
    U에서 w를 추출
      if¬done(w) then ..............................②
        done(w):=true
        w를 D에 추가
        ProcessD
end process
```

ProcessU(t) 프로세스는 RT-Explorer에서 현재노드인 t를 입력받아, 모든 후속노드를 탐색했지만 값이 결정되지 않은 노드들의 값을 결정한다. 참조변수 U의 첫번째 노드의 블럭과 현재노드 t의 블럭이 일치하고 U가 비어있지 않다면 ① 이하를 반복한다. U에 있는 노드 w를 추출하여 ②를 수행한다. ②에서는 done(w)가 거짓이라면 done(w)의 값을 참으로 설정하고 w를 D에 추가하여 ProcessD를 수행한 후을 반복한다.

(7) 검사
기본적으로 지역모형검사에서는 초기노드의 값에 초점을 두기 때문에, (그림 7)에서처럼 RPG를 입력으로 하여 초기노드의 value의 값이 참인지 거짓인지에 따라 시간 오토마타 명세 모델이 검증 항목인 실시간 안전성 및 실시간 필연성을 만족하는지를 확인한다. 자세한 검증 결과는 실시간 안전성 검사 및 실시간 필연성 검사를 통해서 확인한다.

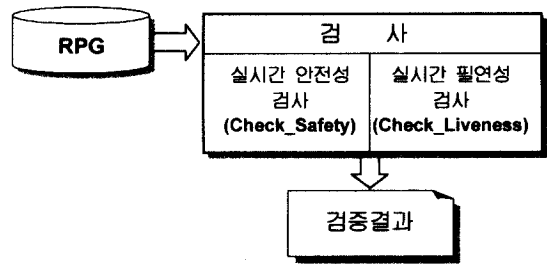



그림 7. RPG 검사 프로세스

Check_Safety 프로세스
(입력) RPG
(출력) result

```
process Check_Safety(RPG)
  RPG의 초기노드를 t0에 입력
  if value(t0)=true then ..............................①
    result := true
  else ......................................................②
    result := false
    for RPG의 모든 노드 t에 대해서then ....③
      if X(t)=6인 value(t)=false then ........④
        if edge(t)가 없고, X(t)=1인 value(t)=false
        then
          S(t)는 deadlock 상태
        else
          S(t)는 livelock 상태
      else if X(t)=3인 value(t)=false then ..⑤
          S(t)는 deadlock이면서, livelock인 상태
  return result
end process
```

Check_Safety(RPG) 프로세스는 RPG를 입력으

로 받아 시간 오토마타 명세 모델이 실시간 안전성을 만족하는지를 확인할 때 사용된다.

RPG의 초기노드를 t0에 입력한 후, ①을 수행하여 value(t0)가 참이면 검증 결과는 참이 되어 프로세스를 종료하고, value(t0)가 거짓이면 ②로 가서 검증 결과는 거짓이 되고 ③을 수행한다. RPG의 모든 노드 t에 대해서 X(t)=6 (X6=<->X2)이고 value(t)=false인 노드를 탐색한다④. 만약 탐색된 노드 중에서 에지가 없고 X(t)=1 (X1=∀X3) 인 value(t)=false라면, S(t)는 실시간 deadlock이 발생하는 상태이고, 그렇지 않을 경우에는 실시간 livelock이 발생하는 상태이다. ⑤는 X(t)=3 (X3=[-]X0)이고 value(t)=false인 노드를 탐색함으로써 ④에서 검출되지 않았던 실시간 deadlock이면서 livelock인 상태 S(t)를 검출한다.

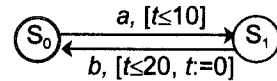
Check_Liveness 프로세스
(입력) RPG, 사용자 입력 (act1, act2, timec)
(출력) result

---

process *Check_Liveness*(RPG)
  for RPG의 모든 노드 *t*에 대해서**then**
    if X(*t*)=4이고 *value*(*t*)=false **then** ·········①
      S(*t*)는 unsafety 상태
    else if X(*t*)=9이고 *value*(*t*)=false **then** ②
      S(*t*)에서 *act1* 발생 후 *act2*가 *timec*를 만족하여 발생할 수 없음
    else if 모든 X(*t*)=9에 대해 *value*(*t*)=true **then**
    ···③
      S(*t*)에서 *act1* 발생 후 *act2*가 *timec*를 만족하여 발생
  return *result*
end process

Check_Liveness(RPG) 프로세스는 실시간 필연성 초기화에서 사용자로부터 입력 받았던 act1, act2, timec와 RPG를 입력으로, 시간 오토마타 명세 모델이 실시간 필연성을 만족하는지를 확인할 때 사용된다. RPG의 모든 노드 t에 대해서 만약 X(t)=4 (X4=[-]X2)이고 value(t)=false인 노드가 검출되면, S(t)는 실시간 안전성을 만족하지 않는 경우이므로 실시간 필연성의 만족여부를 확인할 수 없다①. 그렇지 않고 X(t)=9 (X9=<act2>X10) 인 value(t)=false라면, 상태 S(t)에서는 행위 act1이 발생한 후, 행위 act2가 시간제약 timec를 만족하는 시간영역 내에서 결코 발생하지 않음을 나타낸다②. 논리변수가 X(t)=9 (X9=<act2>X10)인 모든 노드에서 인 value(t)=true라면, 상태 S(t)에서는 행위 act1이 발생한 후, 행위 act2가 시간제약 timec를 만족하는 시간영역 내에서 발생함을 나타낸다③.

3.4.2 실시간 지역모형검사 적용 예

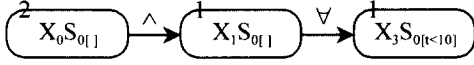
시간 오토마타로 명세한 실시간 시스템은 실시간 지역모형검사 기법을 사용하여 논리적 정확성뿐만 아니라 시간적 정확성을 검증할 수 있다. 본 소절에서는 실시간 지역모형검사가 시간의 무결성을 입증할 수 있음을 예를 통하여 증명한다. (그림 8)은 시간 오토마타로 명세한 실시간 시스템 모델 (a)와 T_μ로 기술된 검증하고자 하는 명제인 논리식 (b)를 *max* 블록과 *min* 블록에 따라 구분하여 나타내고 있다.

(a) 시간 오토마타 명세 모델

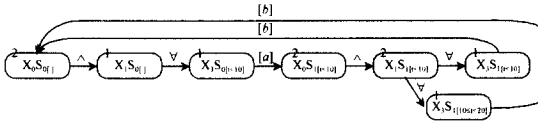
$B_{max} \equiv max\{X_0 = X_1 \wedge X_2$ $X_1 = \forall X_3$ $X_3 = [-]X_0\ \}$	$B_{min} \equiv min\{X_2 = X_4 \vee X_5$ $X_4 = p$ $X_5 = \exists X_6$ $X_6 = \langle - \rangle X_2\ \}$
$\nu Y.\forall[-]Y \wedge (\mu X.A \vee \exists\langle - \rangle X)$	

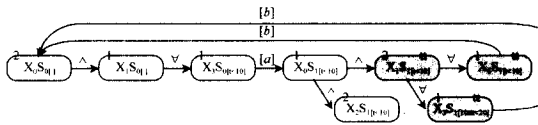
(b) Timed mu-calculus 논리식

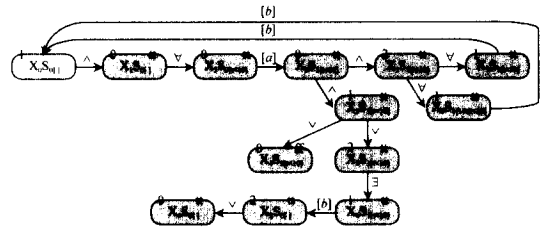
그림 8. 실시간 시스템 모델과 명제 논리식

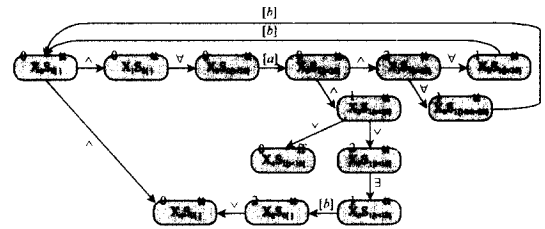
(b)의 논리식을 살펴보면, 시스템 모델 (a)의 안전성을 검증하기 위해서 deadlock이나 livelock이 있는지를 행위 측면과 시간 측면을 모두 고려하여 확인하고자 하는 실시간 안전성 논리식이다. 따라서 실시간 지역모형검사를 수행하여 시스템 모델 (a)가 (b)에서 제시한 실시간 안전성을 만족하는지를 확인한다. 노드의 좌측 상단에는 counter, 우측 상단에는 결정된 value, 노드의 색깔은 done을 나타내며, 에지의 라벨은 노드와 후속 노드간의 연산 관계를 나타낸다.

[단계 1] 초기 노드 $<X_0,S_0[]>$를 생성하여 후속노드를 찾아 에지를 구성하고 $<X_1,S_0[]>$가 시제연산자이므로 시간영역 규칙을 적용하여 제약된 시간영역을 가지는 후속노드 $<X_1,S_0[t<10]>$를 생성한다.

V = $\{<X_0,S_0[]>\ <X_1,S_0[]>\ <X_3,S_0[t<10]>\}$
N = $\{<X_0,S_0[]>\ <X_1,S_0[]>\ <X_3,S_0[t<10]>\}$
D = { }, U = { }

[단계 2] $<X_1,S_0[t<10]>$가 행위연산자이므로 시간영역 규칙을 적용하여 천이가 가능한 a를 찾아 에지를 구성한다. $<X_3,S_1[t<10]>$과 $<X_3,S_1[10\le t<20]>$는 b에 의한 천이가 발생할 때 클럭 t가 재설정되기 때문에 이미 구성된 노드인 $<X_0,S_0[]>$로 에지를 구성한다. $<X_1,S_1[t<10]>$, $<X_3,S_1[t<10]>$, $<X_3,S_1[10\le t<20]>$는 후속노드가 모두 탐색 되었으므로 N에서 U로 옮긴다.

V={ ... $<X_0,S_1[t<10]>\ <X_1,S_1[t<10]>\ <X_3,S_1[t<10]>$
$<X_3,S_1[10\le t<20]>$}
N={ ... $<X_0,S_1[t<10]>\ <X_1,S_1[t<10]>\ <X_3,S_1[t<10]>$
$<X_3,S_1[10\le t<20]>$}
D={ }, U={$<X_3,S_1[t<10]>\ <X_3,S_1[10\le t<20]>$
$<X_1,S_1[t<10]>$}

[단계 3] 현재노드인 $<X_0,S_1[t<10]>$과 N의 첫번째 노드인 $<X_2,S_1[t<10]>$은 다른 블록에 속하는 노드이므로 *ProcessU*($<X_0,S_1[t<10]>$)를 수행하여 U에 있는 노드의 초기치로 값을 결정하고 각각에 대한 *ProcessD*를 수행한다.

V={ ... $<X_0,S_1[t<10]>\ <X_1,S_1[t<10]>\ <X_3,S_1[t<10]>$
$<X_3,S_1[10\le t<20]>\ <X_2,S_1[t<10]>$}
N={ ... $<X_0,S_1[t<10]>\ <X_2,S_1[t<10]>$}
D={$<X_3,S_1[t<10]>\ <X_3,S_1[10\le t<20]>\ <X_1,S_1[t<10]>$}
U={$<X_3,S_1[t<10]>\ <X_3,S_1[10\le t<20]>\ <X_1,S_1[t<10]>$}

[단계 4] $<X_4,S_1[t<10]>$의 값이 거짓으로 결정되어 *ProcessD*를 수행하지만 전속노드인 $<X_2,S_1[t<10]>$이 "∨-node" 이므로 counter를 1 감소하여 또다른 후속노드인 $<X_5,S_1[t<10]>$을 찾아 에지를 구성한다. $<X_5,S_1[t<10]>$의 후속노드인 $<X_6,S_1[t<10]>$을 찾아 에지를 구성하고, 그 후속노드인 $<X_2,S_0[]>$에서 $<X_4,S_0[]>$로 에지를 연결시킨다. $<X_4,S_0[]>$가 극소명제 p를 만족하므로 값을 참으로 결정하여 *ProcessD*를 수행하게되면, 그 전속노드들인 $<X_5,S_1[t<10]>$, $<X_2,S_1[t<10]>$, $<X_0,S_1[t<10]>$, $<X_3,S_0[t<10]>$, $<X_1,S_0[]>$의 counter도 모두 0가 되어 값은 참으로 결정된다.

V={ ... $<X_4,S_1[t<10]>\ <X_5,S_1[t<10]>\ <X_6,S_1[t<10]>$
$<X_2,S_0[]>\ <X_4,S_0[]>$}
N={$<X_0,S_0[]>\ <X_1,S_0[]>\ <X_3,S_0[t<10]>\ <X_0,S_1[t<10]>$
$<X_2,S_1[t<10]>\ <X_5,S_1[t<10]>\ <X_6,S_1[t<10]>\ <X_2,S_0[]>$
$<X_4,S_0[]>$}
D={$<X_4,S_1[t<10]>\ <X_4,S_0[]>\ <X_2,S_0[]>\ <X_6,S_1[t<10]>$
$<X_5,S_1[t<10]>\ <X_2,S_1[t<10]>\ <X_0,S_1[t<10]>$
$<X_3,S_0[t<10]>\ <X_1,S_0[]>$}
U={ }

[단계 5] $<X_0,S_0[]>$의 또 다른 후속노드인 $<X_2,S_0[]>$는 이미 탐색 되었으며 값이 참으로 결정되었기 때문에 counter는 0이 되어 $<X_0,S_0[]>$의 최종값은 참이 되고 전체 과정을 종료한다.

V={$<X_0,S_0[]>\ <X_1,S_0[]>\ <X_3,S_0[t<10]>\ <X_0,S_1[t<10]>$
$<X_1,S_1[t<10]>\ <X_3,S_1[t<10]>\ <X_3,S_1[10\le t<20]>$
$<X_2,S_1[t<10]>\ <X_4,S_1[t<10]>\ <X_5,S_1[t<10]>$
$<X_6,S_1[t<10]>\ <X_2,S_0[]>\ <X_4,S_0[]>$}
N={$<X_0,S_0[]>$}

D={ }, U={ }

위 예에서 보듯이 초기노드의 값이 참이므로 (그림 8. a)의 시스템 명세 모델은 (b)의 실시간 안전성을 만족함을 알 수 있다.

4. 결 론

본 논문에서 제시된 실시간 지역모형검사는 형식 검증 기법에 기반을 두며, 시스템의 행위 측면을 시간 오토마타로 기술한 시스템 모델이 Timed mu-calculus로 표현된 시스템의 특성을 만족하는지의 여부를 실시간 지역모형검사 기법 제시를 통해 시스템 명세의 완전성을 확인하였다. 또, 비록 전역 모형 검사에 비해 과정이 다소 복잡하지만 불필요한 노드 구성과 노드 분할을 억제함으로써 검증 시간을 단축하고 컴퓨터 자원을 절약할 수 있었다.

멀티미디어 서비스 및 실시간 시스템 구현 과정에서 실시간적 요구사항을 시간 오토마타로 명세화하고 명세화를 거친 명세 모델이 요구사항과 일치하는지 구현전에 확인하는 실시간 검증기 구현에 본 논문에서 제시된 알고리즘과 방법론을 적용하면 실시간 검증에서 야기되는 상태폭발문제를 해결할 수 있어 효과적인 실시간 검증기 구현이 가능하다. 그러나 실시간 프로토콜 시스템의 크기가 너무 클 경우에는 제안된 지역모형검사만으로는 상태폭발문제를 해결하기에는 역부족이다. 따라서 제안된 기법과 함께 원 시스템의 상태 크기를 추상화/정제화[10] 시킬 수 있는 연구가 병행되어야 한다.

참 고 문 헌

[1] Kenneth L. McMillan, Symbolic model checking, Kluwer Acacenic Publishers, Model checking, 1996.

[2] P. V. Koppol and K. C. Tai, "Conformance Testing of Protocol specification as Labeled Transition system", International Workshop on Protocol Test System, IWPTS'95, pp.143-158, Evry, France, September, 1995.

[3] 박용범, 김태균, 김성운, "LTS 명세 검증을 위한 모델 검증기 개발" 한국정보처리학회논문지 5권4호, 1998.4

[4] R. Cleaveland, M. Dreimuller and B.steffen, "Faster Model-checking for the Modal Mu-Calculus.", To appear in Proceedings of the 1992 Workshop on Computer-Aided Verification, Lecture Notes in Computer Science.

[5] Emerson, E.A. and C.L. Lei, "Efficient Model Checking in Fragments of the Propositional Mu-Calculus.", Proceedings of the First Symposium on Logic in Cimputer Science, 267-278, 1986.

[6] R. Alur and D. L. Dill. "The theory of timed automata", Theoretical Comput. Sci., 126(2), 1994.

[7] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. "Symbolic model checking for real-time systems", Information and Computation, 111(2), 1994

[8] R. Alur and T. A. Henzinger. Logics and models of real time: A survey. In Real-time: Theory in Practice. LNCS 600, 1991.

[9] Oleg Sokolsky, "Efficient Graph-based Algorithms for Model Checking in the Modal mu-calculus", Ph.D Thesis, Univ. of New York, May., 1996.

[10] Abelardo Pardo, Gary D. Hachtel, "Incremental CTL Model Checking Using BDD Subsetting", Proceedings of the 35th annual conference on Design automation conference, pp.457-462, 1998

박 재 호

1998년 2월 부경대학교 정보통신공학과 학사
1998년 3월~현재 부경대학교 정보통신공학과 석사과정
관심분야 : 검증, ATM, Optical Network

김 성 길

1996년 2월 부경대학교 정보통신공학과 부전공 이수
1998년 9월~현재 부경대학교 정보통신공학과 석사과정
관심분야 : ATM, Optical Network, 실시간 시스템

황 선 호

1982년 9월~1985년 2월 연세대학교 공과대학 전자공학 석사
1992년 8월~1997년 8월 연세대학교 공과대학 통신공학 박사
1984년 3월~1990년 3월 한국전자통신연구원 연구원
1990년 3월~1997년 12월 한국전자통신연구원 선임연구원
1998년 3월~1998년 8월 한국전파기지국관리(주) 연구소장
관심분야 : 이동통신, 무선망, IMT 2000, RF시스템

김 성 운

90년과 93년에 프랑스 파리 7대학 석사 및 박사학위를 취득하였다. 82년부터 85년까지 한국전자통신연구원에서 근무하였으며, 86년부터 95년까지 한국통신연구개발원에서 근무하였음. 96년 이후 부경대학교 정보통신공학과 조교수로 재직중임.
관심분야 : 초고속망 및 프로토콜공학분야임.